APPENDIX B

A copy of an article appearing on pages 1651-1661 of the IEEE Journal of Solid-State Circuits, Vol. 26, No. 11, November 1991 is enclosed as an example of monotonic function blocks.

A Zero-Overhead Self-Timed 160-ns 54-b CMOS Divider

Ted E. Williams, *Student Member, IEEE*, and Mark A. Horowitz, *Member, IEEE*

Abstract —**This paper describes the design of a custom integrated circuit for the arithmetic operation of division. The chip uses self-timing to avoid the need for high-speed clocks, and directly concatenates precharged function blocks without latches. Internal stages form a ring that cycles without any external signaling. The self-timed control introduces no serial overhead, making the total chip latency equal to just the combinational logic delays of the data elements. The ring's data path uses embedded completion encoding and generates the mantissa of a 54-b (floating-point IEEE double-precision) result. Fabricated in 1.2-μm CMOS, the ring occupies 7 mm^2 and generates a quotient and done indication in 45 to 160 ns, depending on the particular data operands.**

I. Introduction

Traditional synchronous circuits separate blocks of combinational logic with clocked latches or registers. As technology improves and logic gets faster, the full utilization of clock periods requires higher clock speeds or the packing of more logic into each clock cycle. However, higher clock speeds introduce additional costs and the need for wider relative margins for clock skew, and computational applications that are sequential in nature may not be as able to take advantage of increased logic complexity within each clock cycle.

The direct algorithms for implementing the arithmetic operation of division exemplify a sequential task. Previous approaches at increasing the performance of division have concentrated on increasing the complexity of the logic function performed in each clock cycle [4], [14]. Higher radix arithmetic can utilize this additional complexity to reduce the number of clock cycles required for a given problem. An alternate approach to avoid constraints from clock speed limitations is to use self-timing [13] to control a sequence of function blocks. Self-timing can attain high performance because sequences of simple steps can be performed without waiting for data to be resynchronized with a global clock at each latch.

The method for self-timing presented in this paper differs from other "self-timed" circuits that are based on matched delays or tuned races [3], [12]. Matching delay elements requires specific process knowledge and extensive engineering and simulation to ensure the designed circuits function correctly over the envisioned ranges of process variance and environmental conditions. In contrast, the method of self-timing demonstrated in this paper constructs circuits that are primarily "speed-independent," meaning that they work correctly for any values of gate delays. The control logic for speed-independent circuits senses when operations are complete, based on information embedded in the data path. But unlike other speed-independent circuits [7], the control logic used here does not enter into the normal critical paths and therefore adds zero overhead to the total delay. The performance obtained is dependent solely on the delays through the combinational data path elements.

Section II of this paper defines the division algorithm that we chose to implement using self-timing. The arithmetic steps required for this algorithm are constructed using the precharged function blocks described in Section III. The design groups these blocks into stages in a configuration, discussed in Section IV, that overlaps the execution of neighboring stages. We arranged the stages in a ring so that the iterative division computation cycles under self-timed control. Like a ring oscillator, this "self-timed ring" operates without external control after initialization with the data operands for a given computation [5]. Section V characterizes, as a function of their local parameters, the performance of the stages when they are connected in an iterating self-timed ring [20]. The data for the division computation must loop around the ring enough times to determine the desired number of quotient digits. As the ring calculates quotient digits, they are captured and accumulated in shift registers. These quotient shift registers are also self-timed and Section VI discusses their structure and a mechanism that can terminate the iterations early and provide a final result sooner in the case of a repeating quotient. Measurements and test results are presented in Section VII from fabricated chips implementing the self-timed divider design. Section VIII compares these results with estimates of other possible self-timed design alternatives and with other published division implementations. Section IX summarizes the performance enhancements for division discussed in this article and the benefits of using a self-timed ring with no latches and zero overhead.

Manuscript received April 15, 1991; revised June 28, 1991.

T. E. Williams is with HaL Computer Systems, Campbell, CA 95008.
M. A. Horowitz is with the Center for Integrated Systems, Stanford University, Stanford, CA 94305.

IEEE Log Number 9102701.

0018-9200/91/1100-1651$01.00 ©1991 IEEE

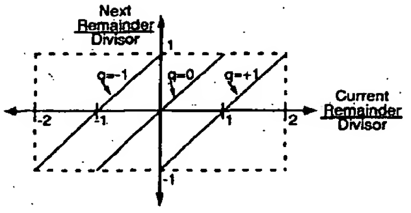

Fig. 1. Radix-2 SRT division diagram.

II. Division Algorithm

Floating-point operations require the computation of both an exponent and mantissa. For division, the exponent of the quotient is just the exponent of the dividend minus the exponent of the divisor, minus one if the mantissa of the dividend is less than the mantissa of the divisor. Since the computation of the exponent is so simple and can be performed in parallel with the mantissa computation, this paper discusses only the mantissa components.

Conventional direct division algorithms determine quotient digits sequentially. A nonrestoring algorithm must choose a valid quotient digit at each stage and update the partial remainder according to the equation:

$$P_{i+1} = rP_i - Dq_i \tag{1}$$

where r is the radix of the algorithm, D is the divisor, P_i is the partial remainder at stage i, and q_i is the quotient digit selected by that stage. The iterations are initialized by setting rP_0 equal to the dividend. Each quotient digit must then be selected so that the next partial remainder stays within specific bounds dependent on the radix [1].

The use of both positive and negative partial remainders and signed quotient digits has become known as SRT division [9]. Signed digits mean a value can be represented in multiple ways, and the arithmetic results are called redundant. Fig. 1 illustrates graphically a stage of a radix-2 SRT division algorithm, which uses quotient digits in the set $\{-1, 0, +1\}$. The key advantage of this scheme is that each stage can select a quotient digit that is valid for a range of values around each possible remainder value. This allows the quotient digit selection to be based on an approximation of the partial remainder in each stage. The approximation can be formed by the most significant bits of the partial remainder, and thus all of the less significant bits can be computed using a carry-save adder (CSA) instead of a full carry-propagate adder (CPA) in each stage. Only a short CPA is needed to combine the sum and carry bits actually examined by the quotient selection logic. The number of bits is dependent on the radix and [15] tabulates several options including the possibility of propagating a different number of sum bits and carry bits from the CSA.

Our estimates for self-timed division designs suggested a radix-2 approach obtains the best performance because its simplicity allows fast stages in a reasonable area. We chose therefore to implement a radix-2 algorithm, but Section VIII will compare in more detail our measured results with other possibilities using higher radixes.

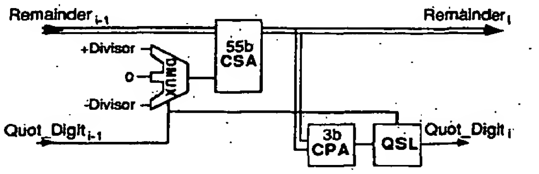

Fig. 2. Simple data flow required for each SRT division stage.

For radix-2 division [6], the CPA must normally examine 4 b even though the constraints of Fig. 1 mean the result is always representable in 3 b. This anomaly occurs because the top bit is needed only due to the possibility that either the sum term or the carry term from the CSA may individually represent a number just slightly more negative than the left boundary of Fig. 1, even though their sum must lie within the boundaries. However, for radix 2, the partial remainder only takes on such an extreme value in the cases for which it was also possible for the preceding stage to select two -1 quotient digits in advance [19]. Therefore, we can narrow the data path by 1 b everywhere, and use only a 3-b CPA, by modifying the standard quotient selection logic equations to

$$q_i = +1 \quad \text{if } \hat{P}_i \geqslant 0 \text{ and } F_{i-1} = 0 \tag{2}$$

$$q_i = 0 \quad \text{if } \hat{P}_i = -1 \text{ and } F_{i-1} = 0 \tag{3}$$

$$q_i = -1 \quad \text{if } \hat{P}_i \leqslant -2 \text{ or } F_{i-1} = 1 \tag{4}$$

$$F_i = 1 \quad \text{if } \hat{P}_i = -4 \text{ or } \left(F_{i-1} = 1 \text{ and } \left[\hat{P}_{i-1}\right]_{msb} = 0\right) \tag{5}$$

where $\hat{P}_i$ is the approximated partial remainder in $\{-4, \cdots, +3\}$ at stage i, and F_i is a flag that is set to force the next quotient digit to be -1. Even when the top sign bit is trimmed away and the remainder "aliases" into a positive number, the quotient selection logic in (2)–(4) always chooses a correct quotient digit. Equation (5) sets the force-next-digit flag when the most negative quotient digit occurs or if the flag was set before and the remainder is already aliased.

Fig. 2 summarizes the hardware requirements for an SRT division stage that uses (2)–(5). Compared with a standard radix-2 algorithm, the reduction in the required size of the remainder data path and adder widths from 4 b to only 3 b improves the total performance by about 5%.

III. Dual-Monotonic Function Blocks and Completion Indication

Circuits can achieve self-timed operation by carefully matching delays between components or by operating control logic with completion information encoded within data signals. Our chip design demonstrates the latter method by using local completion detectors and hand-

TABLE I
ENCODINGS ON A DUAL-MONOTONIC WIRE PAIR

Wire A^T	Wire A^F	Signal A
0	0	Reset = Not Ready
0	1	Evaluated FALSE
1	0	Evaluated TRUE
1	1	Not used = Never occurs

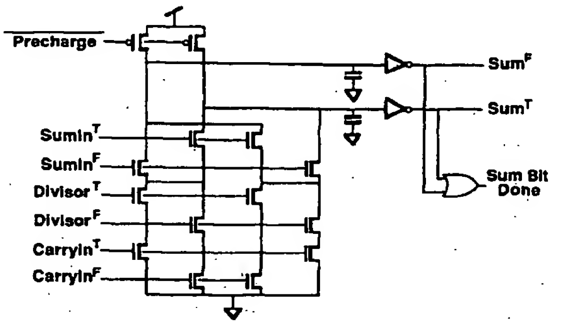

Fig. 3. Precharged dual-rail carry-save adder function block with merged pull-down tree and completion detector.

shaking between fully asynchronous blocks. This approach allows the design to function correctly for any values of gate delays. This insensitivity to component delays makes the design robust because its proper operation is not dependent on detailed process-specific delay information or calculations.

Completion information is embedded in the data throughout the design by using a pair of wires for each bit. Called a "dual-monotonic pair," the wires transmit both a value and a timing signal by using the protocol in Table I. Precharged function blocks can conveniently use dual-monotonic signals as inputs and generate them as outputs. N-channel pull-down networks choose which of the wires in each pair to set high when input data arrive. The pull-down trees for the two output wires can be merged together to share transistors. Fig. 3 shows an example of a function block with a merged pull-down tree that computes the sum bit outputs in a bit slice of an adder used for computing partial remainders. The block takes inputs on three dual-monotonic pairs and outputs the sum bit on another dual-monotonic pair. When one wire in all three inputs goes high, there is a path through the merged pull-down network that discharges one of the two precharged nodes, causing one of the two output wires to rise.

The function block in Fig. 3 precharges when the precharge input is asserted and at least one of the dual-monotonic input pairs is reset. Alternatively, the function block can reset without requiring an input pair to be reset if the transistor stacks are ratioed or if a series n-channel transistor is added to the pull-down network and connected to the precharge input.

A completion signal for a block producing a dual-monotonic pair output can be generated by an OR gate on the output wires, as also shown in Fig. 3. For a fully speed-independent circuit, the individual completion signals from all the bits in a data path must be combined together using a tree of C-elements (a C-element is a gate that sets its output equal to its inputs when the inputs are the same, and otherwise holds its output unchanged). The tree of C-elements can optionally be reduced or omitted in specific cases where it is reasonable to assume that the skew between bits is less than the delay of the OR gate used to detect the completion of each bit. Such an assumption is justified in the portions, or fields, of a data path where the bit slices are all computed in parallel by circuits that do not have a serial dependency and, because of their symmetry, have similar delays for both transition polarities. The assumption is therefore quite valid for the dual-monotonic pairs in a CSA (as in Fig. 3), but a CPA would require the instantiation of a complete tree of C-elements that examines all of the adder's output bits.

An appropriate completion signal can be used by control logic to generate precharge signals for other function blocks. For example, in a simple "domino chain" of blocks, the precharge control signal for each block can be the completion signal from the following block. This connection will precharge each block only after its outputs have been consumed by the following block. Since a block can evaluate only when its precharge signal is removed, the control connection also enforces the symmetric condition that each block can only evaluate when its successor is reset. This control connection therefore checks for the completion of both evaluation and reset operations and ensures that data do not race ahead to corrupt other data values. Other control connections are also possible, with differing performance characteristics [17], [20].

The control logic can also use completion signals to determine when a sequence of operations is finished. The division chip asserts a chip *Done* signal when it completes an entire division computation. This signal can be used by the environment in which the chip is embedded. If the chip is connected to other asynchronous components, the *Done* signal can be used as a *Go* request to the component using the results. If the chip is embedded in a synchronous system, the chip *Done* signal can be used to indicate on which clock cycle the system may take the outputs from the self-timed chip, or to stretch clock cycles as in [21].

IV. DYNAMIC OVERLAPPED EXECUTION STAGES

Our design groups the precharged function blocks implementing the different steps of the division algorithm into stages. Stages having a purely sequential data flow could just directly assemble the full CSA, short CPA, and quotient selection logic (QSL), as shown in Fig. 2. However, the execution of adjacent stages can be overlapped by modifying the stages to allow some parallelism in the data flow. Replicating the CPA's for each possible quotient digit allows each CPA to begin operation before the actual quotient digit arrives at a multiplexor to choose the

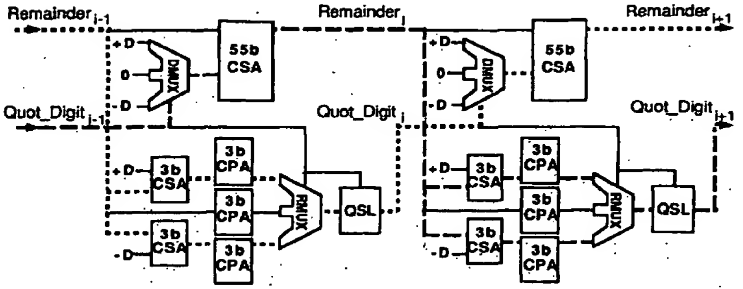

Fig. 4. Data flow through a pair of stages with overlapped execution, showing the two symmetric critical paths.

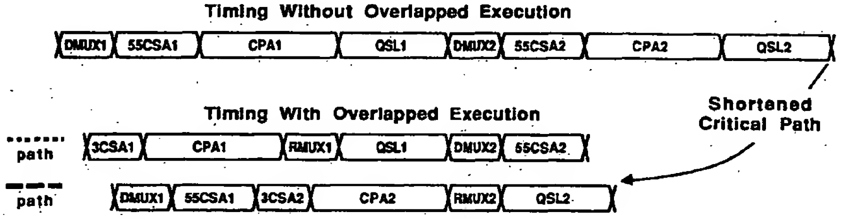

Fig. 5. Comparison of critical paths through a pair of stages, with and without overlapped execution.

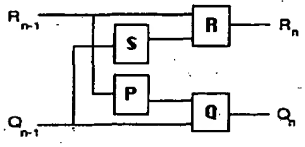

Fig. 6. Model for overlapped execution data flow in each stage.

correct branch. For our radix-2 implementation, we therefore use three CPA's in each stage and a multiplexer (RMUX) to choose between their outputs. Fig. 4 shows the concatenation of the data elements for any two adjoining stages. Two of the three CPA's in each stage are also preceded by CSA's to combine the remainder, respectively, with the divisor and the negation of the divisor. Actually, these two CSA's share the logic for the *sum* terms because the dual-monotonic data convention already provides both the true and complement of each bit. The CSA's cannot share the *carry* terms. Even though the arm for a zero quotient digit still requires a CPA to combine the sum and carry terms of the redundant representation of the partial remainder, the zero arm requires no preceding CSA.

The overlapping of execution between neighboring stages allows the delay through a stage to be the average rather than the sum of the propagation delays through the remainder and quotient digit selection paths. This effect is illustrated by the paths through pairs of stages highlighted with the dashed and dotted lines in Fig. 4. The timing through these two paths is compared with the normal case without overlapped execution in Fig. 5. In order to quantitatively express the benefit of overlapped execution, the arrangement of blocks in the division stages can be abstracted to the model shown in Fig. 6. When these blocks are self-timed and therefore operate as soon as their required operands arrive, the average delay per stage[1] in a chain of identical stages of the overlapped arrangement is

$$\frac{1}{2}\{P+Q+R+S+\max[0,\text{abs}(R-Q)-(P+S)]\} \quad (6)$$

where P, Q, R, and S are the abstracted delays of the blocks. The last term is usually negative and drops out, giving a performance increase due to the factor of 1/2 in front. In the overlapping of the stages for SRT division, the delay of block P in the quotient selection path is the largest because it contains the CPA's. The overlapping reduces by one-half the effect of the delay in block P on the total delay. When the added RMUX delays and the

[1] An exact analysis using induction on the delays of Fig. 6 shows that the time the nth R block finishes in a chain of identical stages is

$$R_n = \frac{n}{2}(P+Q+R+S) + \max\Big[S + \frac{n}{2}(R-S-P-Q), \left(\frac{n}{2}-1\right)(Q-P-R-S), \frac{e}{2}(Q-P-R-S), R-P-Q+\frac{e}{2}(Q+P-R+S)\Big]$$

when the chain's inputs start at $R_0 = Q_0 = 0$ and where

$$e \equiv \begin{Bmatrix} 1 \text{ if } n \text{ odd} \\ 0 \text{ if } n \text{ even} \end{Bmatrix}.$$

Symmetrically, the nth Q block finishes at time

$$Q_n = \frac{n}{2}(P+Q+R+S) + \max\Big[P + \frac{n}{2}(Q-P-R-S), \left(\frac{n}{2}-1\right)(R-S-P-Q), \frac{e}{2}(R-S-P-Q), Q-R-S+\frac{e}{2}(R+S+P-Q)\Big].$$

increased fan-out to the replicated adders are taken into account, the overlapping of the stages in a radix-2 divider design increases performance by 40% over a standard sequential arrangement of the same blocks.

The structure of our overlapping scheme results in a data wavefront that leapfrogs down the succession of stages. If the critical path goes through the quotient selection path in one stage, it will likely go through the partial remainder path in the next stage, and vice-versa. However, data-dependent variances in delays make it possible for the overall minimal critical path to go through the same path in two adjacent stages. Delay variances arise because of the varying number of bits propagated in the carry chains, the occurrence of some zero quotient digits, and the cases in which a negative quotient digit can be selected in advance when a single stage can determine two quotient digits. The self-timing of the data path dynamically ensures data always flow through the minimal critical path.

The probabilistic distribution of quotient digits in SRT division is not uniform due to the asymmetry of the two's complement number system [19]. The asymmetric distribution of quotient digits makes it beneficial to speed up the more frequently used circuit paths since a self-timed implementation can take advantage of the improvement. Our circuit design therefore used different sized transistors in the replicated short CPA's. We designed the $q_i = -1$ adder arm with larger transistors because the critical path goes through it most frequently. We did not use larger transistors in the less frequently chosen $q_i = +1$ arm because the additional loading on the wires also going to the $q_i = -1$ arm would have actually decreased the total performance. The effect of this transistor optimization decreases the probabilistic *expected value* of the delay of driving these adder inputs by about 30%, resulting in an additional 4% improvement in the total divider performance.

V. Self-Timed Ring Structure

The lowest possible latency for an algorithm can be defined as the delay of the combinational array of stages that implements the algorithm. However, a full combinational array has a large area and poor transistor utilization because only a small number of transistors are active at any instant. For an algorithm like SRT division that is repetitive in structure, iterating the computation around a partial array allows the stages to be reused. Previous partial array implementations have required the addition of latches that increased the total latency. However, if the reset signal for each precharged function block is controlled in such a way that the block does not precharge until its results have been consumed and are no longer needed, then it is not necessary to add any explicit latches. Our chip design implements this new idea in order to make an iterating ring that has the same minimal latency as a full array, but occupies only the area of a much smaller partial array.

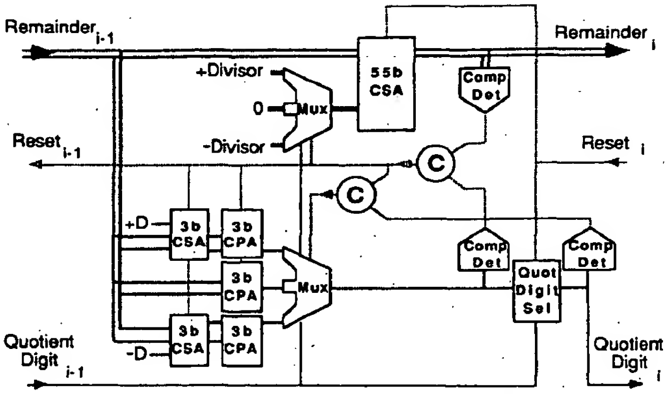

Fig. 7. Internal structure of each stage in the ring implementing an SRT division step with overlapped execution and self-timed reset (precharge) control. Dotted lines show control signals; shaded lines are dual-montonic data paths.

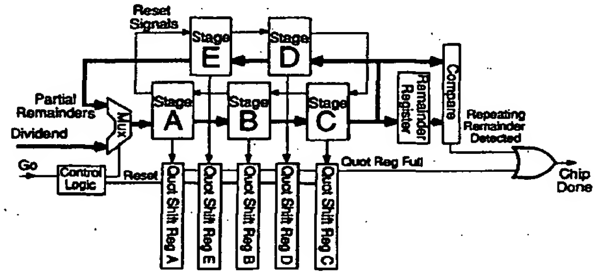

Fig. 8. Block diagram for the division circuit using a ring of domino stages that iterates using self-timing.

The self-timed control logic shown in Fig. 7 precharges a stage's output blocks when their successors have used the data and enables blocks for evaluation when their successors have reset. The control logic uses C-elements to combine the completion-detector outputs from the individual blocks. Using a C-element verifies both the completion of evaluation and the completion of resetting of each block. When stages with this control structure are connected together into a ring as suggested in Fig. 8, the local handshaking allows execution to flow around the ring without being limited by any external signals. The pattern of execution for a five-stage ring is illustrated in Fig. 9.

The self-timed ring also includes some initialization logic and a multiplexor, shown at the left of Fig. 8, that introduces the dividend as the first partial remainder. Control logic switches the multiplexor to close the loop around the ring once a computation has been "launched" into the ring. At the completion of a given problem, initialization logic switches the multiplexor to accept a new dividend for the next computation. Since the divisor remains unchanged throughout each computation, it is held in static registers not shown in the figure.

The performance in a ring can be limited either by the rate at which data elements flow forward or the rate at

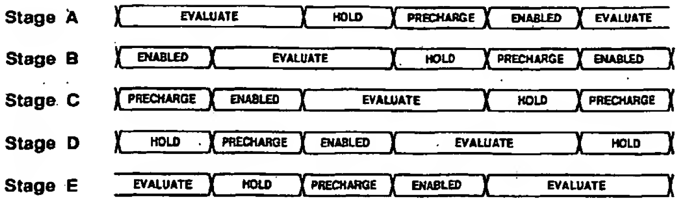
Stage A EVALUATE HOLD PRECHARGE ENABLED EVALUATE
Stage B ENABLED EVALUATE HOLD PRECHARGE ENABLED
Stage C PRECHARGE ENABLED EVALUATE HOLD PRECHARGE
Stage D HOLD PRECHARGE ENABLED EVALUATE HOLD
Stage E EVALUATE HOLD PRECHARGE ENABLED EVALUATE

Fig. 9. Progression of actions through the stages in a self-timed domino ring with five stages. The edges are only approximate since the local handshaking signals are not synchronous to a common clock.

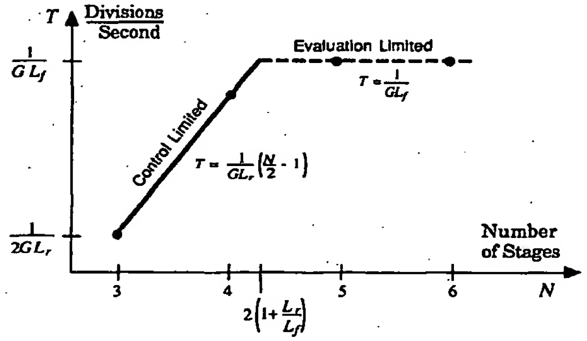

Fig. 10. Performance versus the number of stages.

which control signals flow backward through the handshaking connections [20]. These two effects can be characterized as local parameters of each stage, which we call the "forward latency" L_f and the "reverse latency" L_r. The forward latency is the time between the transitioning of one stage's outputs and the transitioning of the following stage's outputs. The reverse latency is the average delay between the control acknowledgment output of one stage and the control acknowledgment output of the preceding stage. The reverse latency can also be described as the rate at which gaps, or "bubbles," in a data stream flow backward. The control logic in the particular stage configuration in Fig. 7 only influences the reverse latency, which means the forward latency is composed solely of the propagation delays through the combinational logic in the main data path. A key reason that the control logic does not increase the delay of the forward path is because the use of dual-monotonic signaling allows the precharge signal for a function block to be removed, enabling the block's evaluation, before data arrive at the inputs to the block. A precharge signal for a block that had single-ended data inputs could not have been removed until valid data stabilized on its inputs.

The overall performance of a ring can be either "evaluation limited" by the forward latency, or "control limited" by the reverse latency. Fig. 10 illustrates these two possibilities in terms of the number N of stages in the ring and the total number of stage evaluations G required to solve a given division problem. For radix-2 division, G is equal to the number of quotient bits required, which is 54. Fig. 10 is a special case of the more general analysis of self-timed rings derived in [20], which also considers rings that contain extra explicit latches and rings that can circulate multiple data tokens simultaneously. Rings composed of stages without any explicit latches must use at least three stages for data to circulate at all because there must be room for one data element, one reset spacer, and one bubble. For the particular stage configuration shown in Fig. 7, the ratio of the reverse latency to the forward latency is about 1.1, meaning that a ring of those stages requires at least 4.2 stages to avoid having precharge or control logic limit its performance. We therefore chose to use five stages to make sure the ring is evaluation limited and not control limited. The difference,

$$N - 2\left(1 + \frac{L_r}{L_f}\right), \qquad (7)$$

is the "delay margin," the amount by which delays could change before any control logic entered into the critical path. If delays changed by more than this margin, which is equal to about 0.8 stage delays in our design, the circuit would still work correctly, but control logic would enter serially into the overall critical path.

For actual values of delay near the nominal values, no control logic enters into the critical path. Data flow forward continually at the same rate they would flow through an "unwrapped" combinational array implementing the same functions. While previous asynchronous approaches [8], [16] have suffered significant delays due to handshaking control, this method of self-timing operates in the region that adds zero serial control overhead [18] to the latency of the raw function computation. Although there is some small additional loading on the signals that the completion detectors examine, this effect is insignificant compared to the delay that would be introduced if any gates or latches serially lengthened a critical path.

The primary performance benefit of using self-timing to control a chain of directly concatenated precharged stages is that the total latency is not increased by the propagation delays of latches or the propagation and setup delays of synchronously clocked registers. If registers were placed between every pair of stages, the added propagation delay would be about 15% of the logic delays in the stages. Moreover, the required setup time and an allowance to tolerate clock skew would add about another 5% to the achievable clock period.

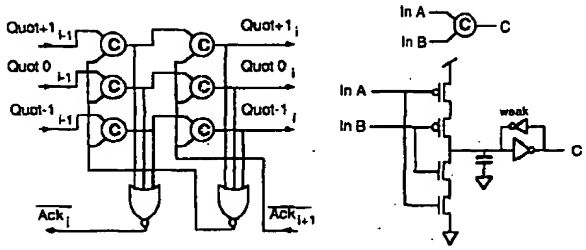

Fig. 11. A cell of the asynchronous shift registers for capturing quotient digits on a triple-monotonic wire set. Each quotient digit arrives when one of the three input wires is set high, and is followed by a "spacer," where all three are again low. A static C-element is defined at right.

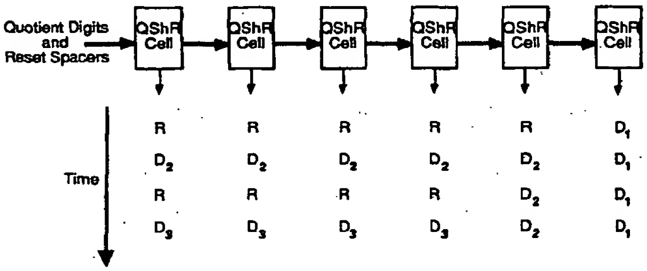

Fig. 12. The asynchronous quotient shift registers pack the digits as they are input, rather than waiting for a fixed number of clocks.

The explicit latch-free self-timed design is also significant in attaining the full benefit of overlapped execution. In the current self-timed design, the different logic delays of the functional blocks skew the remainder and quotient digits passed between the stages by about 10% of the delay through a pair of stages. If any latch or register were inserted between pairs of stages, it would have to wait for the later of the data signals, meaning another 10% degradation would occur because of the data skew.

Together, the degradations of 15%, 5%, and 10% caused by additional propagation delays, clock skew, and data skew, respectively, would add together to produce a total increase in delay of about 30% if latches or registers were placed into the design.

VI. Quotient Accumulation and Early-Done Detection

As the QSL in the five stages of the self-timed ring generates quotient digits, the digits are collected by five separate asynchronous shift registers composed of the cells shown in Fig. 11. Each quotient digit propagates down a chain of shift-register cells to the position from which it will be later read out in parallel, without waiting for a fixed number of clocks as would a synchronous shift register. Fig. 12 illustrates how each digit fills all of the cells through which it propagates with its value. Between each quotient digit sent into a shift register, a reset spacer is sent to separate the digits and to prepare the shift register to accept the next digit. As the reset spacer propagates down the chain, it erases all of the extra copies of the previous quotient digit.

Since the mantissas from floating-point operands are already normalized, the first quotient digit, q_0, is always $+1$. The division ring loops a maximum of 11 times to fill the five shift registers with the rest of the quotient digits up to the total of 54 b required for a double-precision result that will be rounded. On each ring iteration, the remainder comparison on the right side of Fig. 8 determines if the partial remainder has remained unchanged during the last iteration:

$$P_{j+5} = P_j \tag{8}$$

where j is the stage at which the partial remainders begin to repeat. If the remainder repeats, then subsequent remainders and quotient digits will also repeat:

$$\begin{gathered} P_{j+10} = P_{j+5} = P_j \\ q_{j+10} = q_{j+5} = q_j. \end{gathered} \tag{9}$$

Since there is no need to compute the repeating digits again, the iterations terminate and the OR gate at the right of Fig. 8 generates the division *Done* signal early. Even when the iterations terminate early, the full quotient is immediately available from the asynchronous shift registers because the repeated digits are already present. This is possible because the reset spacers between quotient digits are only sent into the shift registers to wipe out the repeated digits after the remainder comparison determines more iterations will be needed. This interlocking of the reset spacers does not add any delay to the overall computation because it occurs in parallel with the evaluation of other quotient digits in the main ring.

The effect on performance of detecting repeating quotients and finishing early is dependent upon the distribution of input operands. Data from some algorithmic applications may be likely to have more round numbers, and data from an external input, like a sensor, may be uniformly distributed only within a limited precision. For example, the early done detection will speed up 12% of the cases in a uniform distribution of 8-b input operands, for a total performance improvement of 9%.

The final quotient can be rounded correctly even when the iterations terminate early. In both the early done and the normal case requiring all of the iterations, the remainder at the stage where the iterations stopped can be sent through a carry-lookahead adder (CLA) to determine its sign. If the remainder is negative, the quotient must be decremented at the least-significant bit position to which the remainder corresponds. This operation and the conversion of the redundant quotient into a standard binary form can be performed by a carry-select adder with multiple carry chains operating in parallel. The different rounding possibilities and the remainder sign select the correct CLA output. Thus, after the iterations terminate, only the delays of a single CLA and multiplexor are required to resolve both the final remainder and rounded quotient.

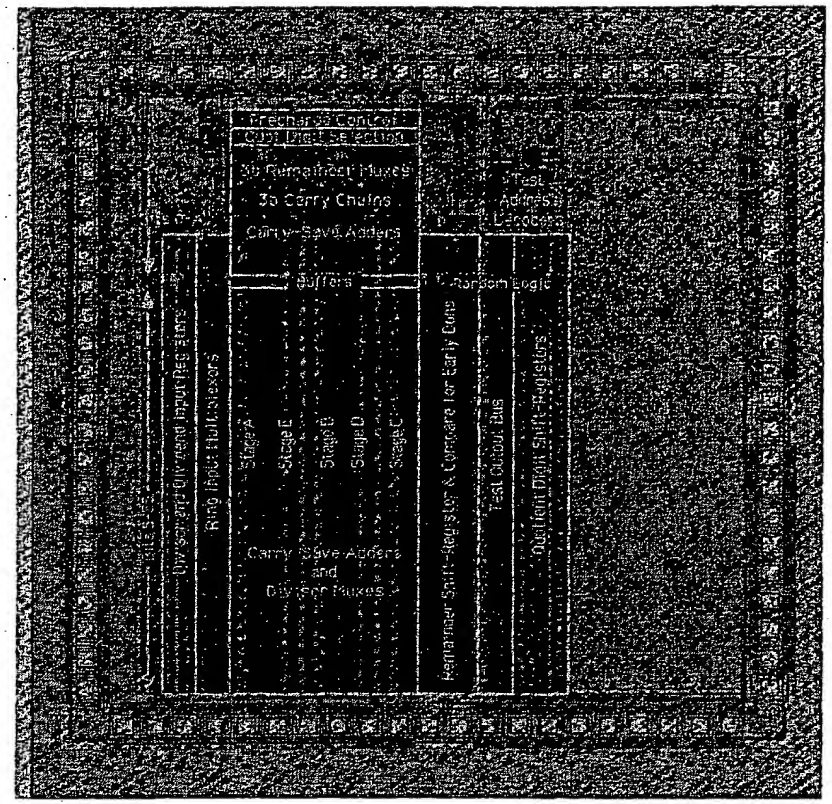

Fig. 13. Micrograph of the zero-overhead self-timed 54-b divider in 1.2-μm CMOS technology.

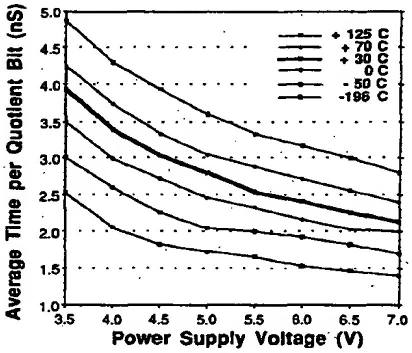

Fig. 14. Measured performance per quotient digit at various voltages and temperatures.

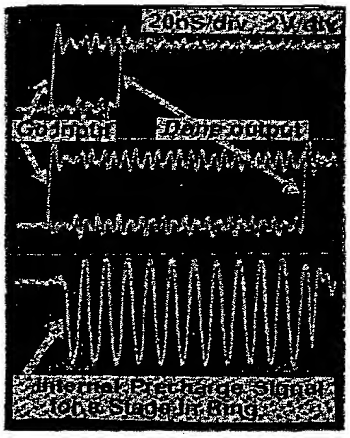

Fig. 15. The top traces show the total latency varies from 45 to 160 ns depending on the data. Below is the self-timed precharge signal for one of the internal stages.

VII. Test Results

We implemented the self-timed divider design using full-custom CMOS circuits. It was important to take special care to minimize charge sharing in the layout of the precharged function blocks by concentrating on minimizing the capacitance of the internal nodes in transistor stacks. We verified the design with IRSIM, a fast event-driven simulator developed at Stanford [11]. This simulator has models accurate enough to allow a designer to predict and correct possible charge sharing hazards. The 45K-transistor chip design was fabricated in 1.2-μm technology at HP through MOSIS, and worked correctly on first silicon.

Our design implements the five stages of the self-timed ring in columns that are mirrored appropriately to weave

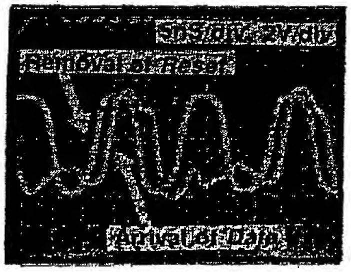

Fig. 16. Superposition of the precharge signal for a domino block with its data input. The precharge signal leads by 2 ns, verifying that it introduces no control overhead.

the data path and achieve equal path lengths. The die photo shown in Fig. 13 shows an active area of 9.7 mm^2, which contains test registers surrounding a core iterating ring in the central 6.8 mm^2. By careful cell design and over-cell routing, the additional area cost occupied by using two wires for each bit was only about 20% of the area required for the transistors. So, even if only single-ended data had been used throughout the design instead of dual-monotonic pairs, the total area would not have decreased by more than about 15%.

The chips generate the correct data outputs over a wide range of operating conditions. The actual operating conditions determine the actual performance, and Fig. 14 shows measured speeds for various voltages and temperatures. For operation at 5 V and 35°C ambient temperature, the ring produces quotient bits internally every 2.8 ns, which is thus the nominal forward latency per stage. Fig. 15 shows that the measured total latency for a 54-b quotient is 160 ns for worst-case data, and 45 ns for best-case data requiring only two ring iterations. This large data dependency in timing shows the effect of the early-done detection.

Since exponent logic could operate in parallel, a complete floating-point division operation could be formed by adding the measured delays for the mantissa operation to only the time required for the single additional 55-b CLA to round and convert the redundant representation into standard binary. In the same technology, this delay would likely be 4 to 8 ns.

Fig. 16 verifies that the design operates without any delay overhead from control logic. It shows that the control logic removes the precharge signals for a block, enabling the block's evaluation, about 2 ns before data circulating around the ring arrive at the block's inputs. This delay margin corresponds to (7) and the choice of using five stages that was suggested by Fig. 10.

VIII. Comparison to Other Alternatives

This section compares the performance of the fabricated design with other possible algorithm options using the same self-timed ring approach, and then concludes with a comparison to commercially available chips for division. Self-timing provides a useful means of comparing the performance of different architectures without constraints from fixed clock cycles, or delays from latches or input/output considerations.

We based estimates for self-timed ring implementations of SRT radix-2 and radix-4 design choices on circuit simulations that have been updated and calibrated with the measurements from the fabricated chips presented in the last section. Table II summarizes the comparisons of these implementation choices, with and without overlapped execution. The delays stated for blocks are specified as multiples of a gate with unity fan-in and fan-out, and include the delays due to buffers on inputs and due to the loading of outputs. The table does not include the final CLA required for rounding and converting the redundant representations back to standard binary. The right two columns contain numbers specific to the CMOS fabrication technology available.

Table II shows that radix 2 is slightly better than radix 4 when neither have overlapped execution. This is because the additional complexity of the radix-4 quotient selection does not quite justify the use of radix 4 when clocking does not need to be considered. However, the difficulty of supplying a clock for a radix-2 design at almost twice the frequency might make radix 4 preferable for clocked designs. Overlapped execution in either radix 2 or radix 4 gives a significant performance increase, about 40% for radix 2 and 55% for radix 4. The key advantage of the self-timed overlapped execution style here is that the average critical path per stage has a factor of 1/2 times the delay from the CPA's and QSL. Since, for higher radixes, these components occupy bigger proportions of the total delay, the effect of overlapped execution is more significant for radix 4. So, although radix 4 is faster than radix 2 when both have overlapped execution, we chose not to implement radix 4 because its area cost in a self-timed ring is much higher due to the replication of the CPA's. Not only are five adders required instead of only three, but they are also larger. Still higher radixes, such as radix 8, would accentuate these trade-off effects. Overlapped execution would have an even greater percentage reduction in delay, but at a formidable cost in area.

The self-timed division design presented in this article is faster than recent commercially available designs. Both the MIPS R3010B chip [10] and the Weitek 3364 chip [2] were built using a 1.2-μm CMOS technology similar to the one available for the self-timed disign. Both of these commercial chips use a synchronously clocked radix-4 division algorithm. The speed for the mantissa portion of a double-precision division is 375 ns for the R3010B chip and 675 ns for the 3364 chip. To be fair, these numbers should be compared to the speed of the self-timed divider chip at the voltage and temperature for which the synchronous chips are specified, even though the self-timed chip indeed can produce its results faster when the actual conditions are not at worst case. From Fig. 9, the self-timed chip's applicable derated speed is 190 ns for a double-precision result, which is still a factor of 2 to 3.5 better than the commercial chips.

TABLE II
TRADE-OFFS IN SPEED AND AREA FOR SELF-TIMED RING IMPLEMENTATIONS OF DIFFERENT ALGORITHMS

Radix & Style (OverExec = Overlapped Execution)	Average Critical Path per Pair of Stages in Unity Fan-in, Unity Fan-out, Gate Delays	Unity Fan-in/out Gate Delays per Quot Bit	Latency for 54 b with 250-ps Unit Gate Delays	Silicon Area in 1.2-μm Technology
Radix 2	2(CPA3 + QSL3 + DMUX3 + CSA55) 2(5.7 + 3.8 + 2.8 + 4.5) = 33.6	16.8	225 ns	7 mm^2
Radix 2, OverExec	CSA3 + CPA3 + RMUX3 + QSL3 + DMUX3 + CSA55 3.9 + 4.9 + 3.5 + 3.8 + 2.8 + 4.7 = 23.6	11.8	160 ns	10 mm^2
Radix 4	2(CPA7 + QSL5 + DMUX5 + CSA56) 2(11 + 16 + 3.5 + 4.5) - 70.0	17.5	235 ns	12 mm^2
Radix 4 OverExec	CSA7 + CPA7 + RMUX5 + QSL5 + DMUX5 + CSA56 3.9 + 10 + 5.0 + 16 + 3.5 + 6.2 - 44.6	11.2	150 ns	18 mm^2

IX. SUMMARY

Self-timed components avoid the need of distributing global clocks and degradations caused by clock skew in synchronous systems. Self-timed circuits that are speed-independent use completion detectors that sense when operations become finished, and use this information to trigger other operations. This approach allows them to work correctly for any values of gate delays. Variances and data dependencies in delays can be used advantageously because each component can begin when its required operands actually arrive rather than always waiting for worst-case timing. Speed-independent circuits are robust because they will continue to function correctly over wide ranges of power supply voltage, die temperature, and fabrication spread.

This paper has presented a novel method of using self-timing to control precharged function blocks that are concatenated together without latches. When the blocks are composed into stages and the stages are arranged in a self-timed ring, the ring can iterate under self-timed control without being limited by the rate of any external signal. For nominal delay values, the control logic never enters into the critical path and the performance is the same as a combinational array implementing the same algorithm, but "wrapped" into a much smaller area.

The performance possible with a latch-free self-timed ring has been demonstrated in a circuit implementing the mantissa computation for floating-point division. The design also has several other performance enhancements, including a symmetric overlapped execution of the stages that allows a dynamically adjusting data-dependent minimal critical path. When fabricated in 1.2-μm CMOS, the chips' measured performance is 160 ns for worst-case data, and 45 ns for round fractions that produce a repeating quotient. Since the design produces a done indication, the outputs can be used as soon as they are available without waiting for worst-case derated specifications.

Self-timed rings can compute iterative functions without limitations from the rate of external signals and without any overhead above the delays of the data elements. The concept is applicable not only to division, but also to square-root, CORDIC algorithms, and other computations in computer arithmetic and signal processing.

ACKNOWLEDGMENT

The authors thank L. Ho and Y. Shimazu for assisting with the chip layout.

REFERENCES

[1] D. Atkins, "Higher-radix division using estimates of the divisor and partial remainder," *IEEE Trans. Comput.*, vol. 17, no. 10, pp. 925–934, Oct. 1968.

[2] M. Birman *et al.*, "Design of a high speed arithmetic datapath," in *Proc. ICCD: VLSI Computers and Processors*, 1988, pp. 214–216.

[3] T. Chappell *et al.*, "A 2ns cycle, 4ns access 512kb CMOS ECL SRAM," in *ISSCC Dig. Tech. Papers*, Feb. 1991, pp. 50–51.

[4] J. Fandrianto, "Algorithm for high speed shared radix 8 division and radix 8 square root," in *Proc. 9th Symp. Computer Arithmetic*, Sept. 1989, pp. 68–75.

[5] M. Greenstreet, T. Williams, and J. Staunstrup, "Self-timed iteration," in *Proc. VLSI-87* (Vancouver, Canada). Amsterdam: Elsevier Science, Aug. 1987, pp. 309–322.

[6] W. McAllister and D. Zuras, "An nMOS 64b floating-point chip set," in *ISSCC Dig. Tech. Papers*, Feb. 1986, pp. 34–35.

[7] A. Martin *et al.*, "The design of an asynchronous microprocessor," in *Proc. Decennial Caltech Conf. Advanced Res. VLSI*, Mar. 1989, pp. 351–373.

[8] T. Meng, R. Brodersen, and D. Messerschmitt, "Automatic synthesis of asynchronous circuits from high-level specifications," *IEEE Trans. Computer-Aided Design*, vol. 8, no. 11, pp. 1185–1205, Nov. 1989.

[9] J. Robertson, "A new class of digital division methods," *IRE Trans. Electron. Comput.*, vol. EC-7, pp. 218–222, 1958.

[10] C. Rowen, M. Johnson, and P. Ries, "The MIPS R3010 floating-point coprocessor," *IEEE Micro*, pp. 53–62, June 1988.

[11] A. Salz and M. Horowitz, "IRSIM: An incremental MOS switch-level simulator," in *Proc. 26th Design Automation Conf.*, June 1989, pp. 173–178.

[12] M. Santoro and M. Horowitz, "SPIM: A pipelined 64×64-bit iterative multiplier," *IEEE J. Solid-State Circuits*, vol. 24, no. 2, pp. 487–493, Apr. 1989.

[13] C. Seitz, "System timing," in *Introduction to VLSI Systems*, C. Mead and L. Conway, Eds. Reading, MA: Addison-Wesley, 1980, ch. 7.

[14] G. Taylor, "Radix 16 SRT dividers with overlapped quotient selection stages," in *Proc. 7th Symp. Computer Arithmetic*, June 1985, pp. 64–71.

[15] T. Williams and M. Horowitz, "SRT division diagrams and their usage in designing custom integrated circuits for division," Stanford Univ., Stanford, CA, Tech. Rep. CSL-TR-87-326, Nov. 1986.

[16] T. Williams *et al.*, "A self-timed chip for division," in *Proc. Stanford Conf. Advanced Res. VLSI*, Mar. 1987, pp. 75–95.

[17] T. Williams, "Latency and throughput tradeoffs in self-timed asynchronous pipelines and rings," Stanford Univ., Stanford, CA, Tech. Rep. CSL-TR-90-431, Aug. 1990.

[18] T. Williams and M. Horowitz, "A zero-overhead self-timed 160ns 54b CMOS divider," in *ISSCC Dig. Tech. Papers*, Feb. 1991, pp. 98–99.

[19] T. Williams and M. Horowitz, "A 160nS CMOS division implementation using self-timing and overlapped SRT stages," in *Proc. 10th Symp. Computer Arithmetic*, June 1991, pp. 210–217.

[20] T. Williams, "Self-timed rings and their application to division," Ph.D. dissertation, Stanford Univ., Stanford, CA, Tech. Rep. CSL-TR-91-482, 1991.

[21] G. Wolrich *et al.*, "A high performance floating-point coprocessor," *IEEE J. Solid-State Circuits*, vol. SC-19, no. 5, pp. 690–696, Oct. 1984.

Ted E. Williams (S'88) was born in Anaheim, CA, in 1962. He received the B.S. degree with Honors from the California Institute of Technology, Pasadena, in 1984. He received the M.S. and Ph.D. degrees in electrical engineering in 1985 and 1991, respectively, from Stanford University, Stanford, CA.

From 1984 to 1986 he was a member of the technical staff at the Xerox and Hewlett-Packard Laboratories, both in Palo Alto, CA. He accepted a position as an Invited Guest Professor from 1986 to 1988 at Aarhus University in Denmark and at the Norwegian Technical Institute in Trondheim, Norway. From 1988 to 1990 he was Vice-President of Integrated Circuit Development at Silicon Engines Inc. He has now joined HaL Computer Systems in Campbell, CA. His current research interests are in self-timed circuits, performance analysis, and computer arithmetic.

Dr. Williams was the Chapter President of Tau Beta Pi and received the Carnation Prize for excellence in academics and research when he was at the California Institute of Technology.

Mark A. Horowitz (S'77–M'83) received the B.S. and M.S. degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, in 1978, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1983.

Currently, he is an Associate Professor in the Department of Electrical Engineering at Stanford University. His research interests are in the area of high-performance digital system design, especially integrated circuit design, and computer tools to aid in the design process. He has worked on a large number of chip designs, ranging from high-speed ECL and BiCMOS RAM's to MIPS-X, a high-speed CMOS RISC microprocessor. His current research areas are new tools and circuits for BiCMOS and high-speed processor design.

Dr. Horowitz is the recipient of a 1985 Presidential Young Investigator Award, and a 1986–1987 IBM Faculty Development Award.